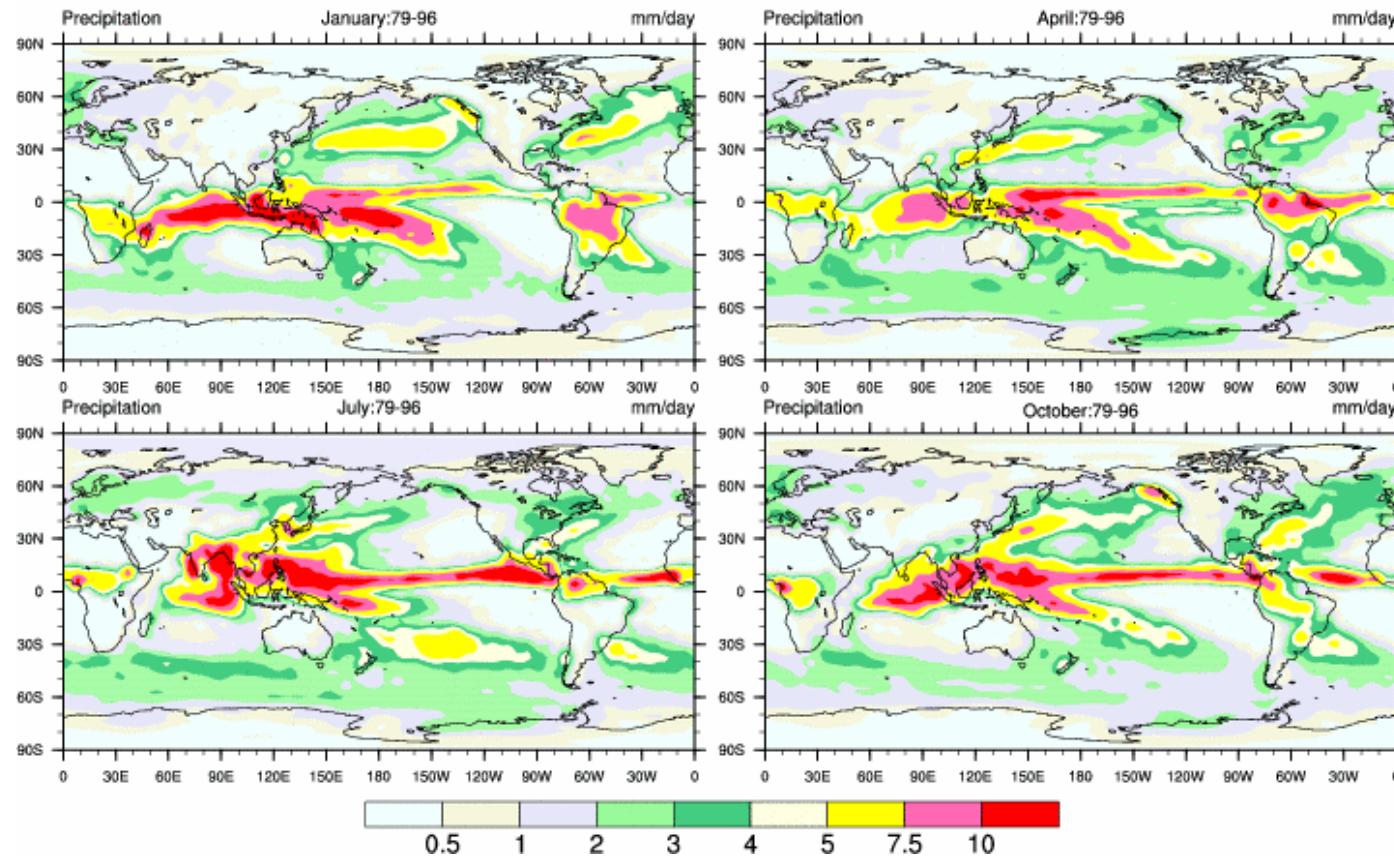


Data Processing with NCL

CPC Merged Prc: Climatology



Dennis Shea

National Center for Atmospheric Research

DP Example: multi-formatted data

- **ISCCP**: HDF, binary: 20+yrs, 3hrly: **80GB** [type byte]
 - **calculations, regrid**, monthly statistics => netCDF, plot
per yr: wc= 25.5h, usr=14.9h, sys=8.1h, **24GB** [total **480GB**]
- **NCEP**: GRIB: (same) 20+yrs, 6hrly: **25+GB**
 - **calculations, regrid**, monthly statistics => netCDF, plot
- **CAM**: netCDF: (same) 20yrs: ?
 - perhaps an ensemble of model runs
 - **calculations, monthly statistics** => netCDF, plot
- **Science**: datasets, calculations, graphics => **paper**

Data Processing Outline

- Algebraic/logical expression operators
- Manual and automatic array creation
- **if** statements
- **do** loops
- Built-in and Contributed functions
- User developed NCL functions/procedures
- User developed external procedures
- Fortran external subroutines
- Examples

Algebraic Operators

Algebraic expression operators

- Negation	^ Exponentiation
*	/ Divide
% Modulus [integers only]	# Matrix Multiply
+	- Minus
> Greater than selection	< Less than selection

- Use (...) to circumvent precedence rules
- All support scalar and array operations [like f90]
- + is overloaded operator
 - algebraic operator:
 5.3 + 7.95  13.25
 - string concatenator:
 "alpha" + (5.3 + 7)  "alpha12.3

Logical Expressions

- Similar to f77/f90

**Logical expressions formed
by relational operators**

- .le. (less-than-or-equal)
- .lt. (less-than)
- .ge. (greater-than-or-equal)
- .gt. (greater-than)
- .ne. (not-equal)
- .eq. (equal)
- .and. (and)
- .xor. (exclusive-or)
- .or. (or)
- .not. (not)

Manual Array Creation

- **array constructor characters (*/.../*)**

- a_integer = *(/1,2,3/)*
- a_float = *(/1.0, 2.0, 3.0/)*, a_double = *(/1., 2, 3.2d0 /)*
- a_string = *(/"a","b","c"/)*
- a_logical = *(/True, False,True/)*
- a_2darray = *(/ (/1,2,3), (/4,5,6), (/7,8,9) /)*

- **new function [Fortran dimension, allocate and C malloc]**

- x = **new** (array_size/shape, type, **missing value**)
 - missing value is **optional** [assigned default if not user specified]
- a = **new**(3, float)
- b = **new**(10, double, **1d+20**)
- c = **new**(*(/5, 6, 7/)*, integer)
- d = **new(dimsizes(U), string)**
- e = **new(dimsizes(ndtooned(U)), logical)**

- **new** and *(/.../)* can appear anywhere in script

- **new** is not used that often

Automatic Array Creation

- **variable to variable assignment**
 - **y = x** **y** => same size, type as **x** plus meta data
 - no need to preallocate space for **y**
- **data importation via supported format**
 - **u = f->U**
 - same for subset of data: **u = f->U(:, 3:9:2, :, 10:20)**
 - meta data (coordinate will reflect subset)
- **functions**
 - return array: **no need** to preallocate space
 - **T42 = f2gsh (gridi, (/ 64,128/), 42)**
 - **gridi(10,30,73,144) ↗ T42(10,30,64,128)**
 - **T42 = f2gsh_Wrap (gridi, (/ 64,128/), 42) ; contributed.ncl**

Array Dimension Reduction

- **let T(12,64,128)**

- $T_{jan} = T(0, :, :)$ $T_{jan}(64, 128)$
- T_{jan} automatically becomes 2D: $T_{jan}(64, 128)$
- array rank has been reduced
- all applicable meta data copied

- **can override dimension reduction**

- $T_{jan} = T(0:0, :, :)$ $T_{jan}(1, 64, 128)$
- $TJAN = \text{new}(\text{/}1, 64, 128\text{/}, \text{typeof}(T), T @_FillValue)$
 $TJAN(0, :, :) = T(0, :, :)$

- **Dimension Reduction is a "feature" [really]**

Array Syntax/Operators

- similar to f90/f95
- arrays must be same size and shape: conform
- let A and B be (10,30,64,128)
 - $C = A+B$
 - $D = A-B$
 - $E = A*B$
 - C, D, E automatically created if they did not previously exist
- let T and P be (10,30,64,128)
 - $\theta = T^*(1000/P)^{0.286}$ ↗ theta(10,30,64,128)
- let SST be (100,72,144) and SICE = -1.8 (scalar)
 - $SST = SST > SICE$ [f90: where (sst.lt.sice) sst = sice]
- use built-in functions whenever possible
 - let T be (10,30,64,128) and P be (30) then
 - $\theta = T^*(1000/\text{conform}(T,P,1))^{0.286}$
- all array operations automatically ignore _FillValue

if statements

- **if-then-end if** (note: **end if** has space)

```
if ( all(a.gt.0.) ) then  
    ...statements  
end if
```

- **if-then-else-end if**

```
if ( any(ismissing(a)) ) then  
    ...statements  
else  
    ...statements  
end if  
no else if
```

- lazy expression evaluation [left-to-right]

```
if ( any(b.lt.0.) .and. all(a.gt.0.) ) then  
    ...statements  
end if
```

loops

- **do loop** (traditional structure; **end do** has space)

- **do i=scalar_start_exp, scalar_end_exp [, scalar_skip_exp]**
do n = 0, N-1 [,stride]
... statements
end do ; 'end do' has a space
 - if start > end
 - ☞ identifier 'n' is decremented by a positive stride
 - ☞ stride must always be present when start>end

- **do while loop**

```
do while (x .gt. 100)
  ... statements
end do
```

- **break:** loop to abort [f90: exit]
- **continue:** proceed to next iteration [f90: cycle]

- **minimize loop usage** (generally)

- use array syntax, built-in functions, procedures
 - use Fortran/C codes when efficient looping is required

Minimize Use of Loops_(1 of 2)

- Loops should be avoided in **any** interpreted language.
- parameters for each example:
 - M=1000, N=1000
 - T=new(/N,M/,float)

```
; 136 seconds

do i= -N/2,(N-1)/2
    do j= -M/2,(M-1)/2
        T(i+N/2,j+M/2) = 100.0 - sqrt((8.0*i)^2 + (8.0*j)^2)
    end do
end do
```

```
; 88 seconds

do i= -N/2,(N-1)/2
    do j= -M/2,(M-1)/2
        T(i+N/2,j+M/2) = i^2 + j^2
    end do
end do
T = 100.0 - sqrt((8.0^2) * T)
```

Minimize Use of Loops (2 of 2)

```
; <1 sec-----  
  
jspn = ispan(-M/2,(M-1)/2,1)^2  
ispn = ispan( -N/2,(N-1)/2,1)^2  
do i = 0,dimsizes(ispn)-1  
    T(i,:) = ispn(i) + jspn  
end do  
T = 100.0 - sqrt((8.0^2) *T )
```

```
; <1 sec  
  
T = onedtond(ispan(-M/2,(M-1)/2,1),(/N,M/))  
T!0 = "x"  
T!1 = "y"  
T = 100.0 - sqrt((8.0*T(:,:,1))^2 + (8.0*T(:,:,2))^2)
```

Built-in Functions and Procedures_(1 of 2)

- **use whenever possible**
- **learn and use utility functions**
 - all, any, conform, ind, ind_resolve, dimsizes
 - fspan, ispan
 - ndtooned, onedtond
 - mask, ismissing
 - system, systemfunc [use local system]
- **functions may require dimension reordering**
 - use named dimensions

;compute zonal and time average of variable T(time,lev,lat,lon)

; dim_avg works on right most dimension

; no meta data transferred

Tzon = **dim_avg(T)** ; Tzon(time,lev,lat)

Tavg = **dim_avg(T(lev|:, lat|:, lon|:, time|:))** ; reorder
; Tavg(lev,lat,lon)

Built-in Functions and Procedures_(2 of 2)

- **functions: NO need to preallocate memory**
 - `y = wgt_runave (x, wgt, 0)`
 - if returning to pre-existing array: must conform
- **procedures: MUST preallocate memory with new**

```
psi = new ( dimsizes(u) , typeof(u) )
chi = new ( dimsizes(u) , typeof(u) )
uv2sfvpg(u,v,psi,chi)
```
- **functions may be imbedded, procedures can not**
 - keep code simple: avoid 'deep' imbedding

```
; example of a deep imbed
x = f2gsh( fo2fsh( fbinrecread(f,6,(/9,18,72,144/), "float")),(/nlat,mlon/),42)
; without deep imbedding
G = fbinrecread(f, 6, (/1,18,72,144/), "float")
g42 = f2gsh( fo2fsh(G), (/nlat,mlon/),42)
delete (G)
```

Type of routines	Description
Array routines	routines for array reshaping, sorting, retrieving the dimension sizes of an array, masking an array, getting bit information, etc.
Climate and model routines	routines for use specifically with climate and model data
Color routines	routines for retrieving color indices of named colors and setting and freeing colors
Computational math routines	routines for computing empirical orthogonal functions, Fourier coefficients, singular value decomposition, cumulative distribution functions, averages, standard deviations, etc.
Conversion functions	functions for converting from one variable type to another
Date routines	routines for retrieving and converting date information
Drawing routines	routines for drawing primitives (lines, filled areas, and markers), wind barbs, weather map symbols, isosurfaces, and graphical objects
File I/O and utility routines	routines for file handling
Interpolation routines	1D, 2D, and 3D interpolation, approximation, and regridding routines
NCL object routines	routines for manipulating and querying NCL objects
NCL variable/procedure routines	routines for creating, deleting, querying, converting, and printing NCL variables, for listing all the variables and procedures, for undefining variables and procedures, etc.
Random number generators	Various random number generators
Regridding routines	Vector and scalar regridding routines
Spherical harmonic routines	routines that facilitate computer analysis of scalar and vector global geophysical quantities (most are based on the package known as Spherepack)
Standard math routines	sin, cosine, log, min, max, etc.
System routines	routines for retrieving environment variables, executing system commands, and returning the path to an NCAR Graphics directory
Workstation routines	routines for changing which workstation the output is drawn to, advancing the frame in a workstation, clearing the workstation, etc.

dimsizes(x)

- returns the dimension sizes of a variable
- will return 1D array of integers if the array queried is multi-dimensional.

```
begin
    fin=addfile("./in.nc","r")
    t =in->T
    dimt = dimsizes(t)
    print(dimt)

    rank = dimsizes(dimt)
    print ("rank="+rank)
end
```

Variable: dimt

Type: integer

Total Size: 16 bytes

4 values

Number of dimensions: 1

Dimensions and sizes:(4)

(0) 12

(1) 25

(2) 116

(3) 100

(0) rank=4

ispan(start:integer, finish:integer, stride:integer)

- returns a 1D array of integers
 - beginning with **start** and ending with **finish**.

```
time = ispan(1990,2001,2)  
print(time)
```

```
time=ispan(1990,2001,2)*1.0  
time would be Type: float
```

Variable: time
Type: integer
Number of Dimensions: 1
Dimensions and sizes:(6)
(0) 1990
(1) 1992
(2) 1994
(3) 1996
(4) 1998
(5) 2000

fspan(start:float, finish:float, num:integer)

- returns a 1D array of evenly spaced floating point numbers
- **num** is the number of points **including start and finish**

```
begin
    b = fspan(1,5,10)
    print(b)
end
```

Variable b:

Type: float

Number of Dimensions: 1

Dimensions and sizes:(10)

(0) 1.0

(1) 1.444

(2) 1.888

(3) 2.333

(4) 2.777

(5) 3.222

(6) 3.666

(7) 4.111

(8) 4.555

(9) 5.0

mask

- sets values to `_FillValue` that **DO NOT** equal mask array.
 - f90: `where(oro.ne.1) ts = ts@_FillValue`

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
begin
```

; read in netCDF file

```
in  = addfile("/wp/cgd/murphys/Data/ATMOS/atmos.nc","r")
ts  = in->TS(0,:,:)
oro = in->ORO(0,:,:)
```

; mask out ocean data

; ocean=0, land=1, sea_ice=2

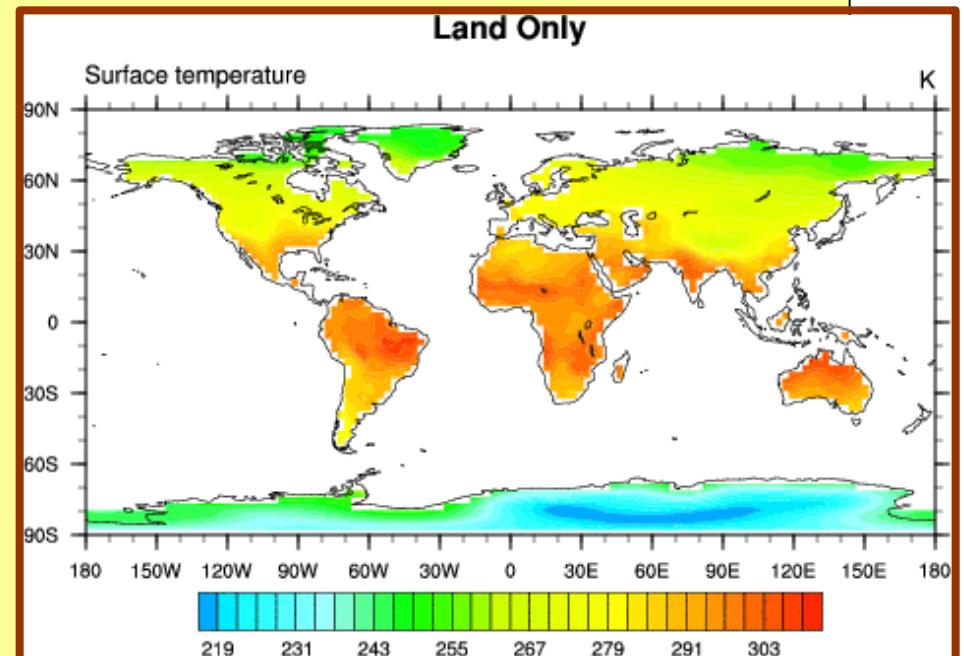
```
land_only  = ts
```

```
land_only  = mask(ts,oro,1)
```

; f90 where(oro.ne.1) \

; land_only = ts@_FillValue

end



ind

- **ind** operates on 1D array only
 - returns indices of elements that evaluate to True
 - if nD ... use with **ndtooned**; reconstruct with **onedtond**, **dimsizes**
 - often used in applications to emulate **f90 where** function

```
; q is 1D  want
; q<0 => q=q+256
; f90: where(q.lt.0)
;       q=q+256
if (any(q.lt.0) ) then
  ii = ind(q.lt.0)
  q(ii) = q(ii)+256
end if
```

```
; let q be an nD array
; f90: where(q.lt.0.) q = q+256
if (any(q.lt.0.) ) then
  q1D    = ndtooned (q)
  ii      = ind(q1D.lt. 0.)
  q1D(ii) = q1D(ii) + 256
  q       = onedtond(q1D, dimsizes(q))
  delete(q1D)      ; no longer needed
end if
```

ind_resolve

- **ind_resolve** will return the nD indices corresponding to **ind**

```
; print indices p > 50 [let p(time,lat,lon)]  
if ( any(p.gt.50.) ) then  
    p1D    = ndtooned (p)  
    i50    = ind(p1D.gt.50.)  
    ir     = ind_resolve( i50, dimsizes(p)) ; 2D [npts,3]  
    print ("ind where p> 50: "+ir(:,0)+" "+ ir(:,1)+" "+ ir(:,2))
```

```
; print time,lat,lon p > 50  
print (time(ir(:,0))+ " + lat(ir(:,1))+ " + lon(ir(:,2)) )  
end if
```

- **CAVEAT:** should **not** be used for vector subscripting

```
u(ir(:,0),ir(:,1),ir(:,2)) = 5 ; won't get what you might expect
```

udunits: ut_calendar, ut_inv_calendar

- <http://ngwww.ucar.edu/ngdoc/ng/ref/ncl/functions/utcal.html>
 - not available under MacOSX or Cygwin ('hidden functions')
 - why? Uduits library not supported under MacOSX/Cygwin

```
time = (/ 17522904, 17522928, 17522952/)
```

```
time@units = "hours since 1-1-1 00:00:0.0"
```

```
date = ut_calendar(time,0)
```

```
print(date)
```

Variable: date

Type: float

Total Size: 72 bytes 18 values

Number of Dimensions: 2

Dimensions and sizes: [3] x [6]

(0,0:5) 2000 1 1 0 0 0

(1,0:5) 2000 1 2 0 0 0

(2,0:5) 2000 1 3 0 0 0

```
TIME = ut_inv_calendar (iyr, imo, iday, ihr, imin, sec \
```

```
, "hours since 1-1-1 00:00:0.0" ,0)
```

udunits: **ut_calendar**, **ut_inv_calendar**

- <http://ngwww.ucar.edu/ngdoc/ng/ref/ncl/functions/utcals.html>

```
time = (/ 17522904, 17522928, 17522952/)  
time@units = "hours since 1-1-1 00:00:0.0"  
date = ut_calendar(time,0)  
print(date)
```

Variable: date

Type: **float**

Total Size: 72 bytes 18 values

Number of Dimensions: 2

Dimensions and sizes: [3] x [6]

(0,0:5) 2000 1 1 0 0 0

(1,0:5) 2000 1 2 0 0 0

(2,0:5) 2000 1 3 0 0 0

```
TIME = ut_inv_calendar (iyr, imo, iday, ihr, imin, sec \  
,"hours since 1-1-1 00:00:0.0" ,0)
```

system, systemfunc

- **system** passes **to** the shell a command to perform an action
- **systemfunc** returns to NCL information **from** the system
- NCL executes the Bourne shell (can be changed)

; create a directory if it does not exist (Bourne shell syntax)

```
DIR = “/ptmp/shea/SAMPLE”
```

```
system(“if ! test -d “+DIR+” ; then mkdir “+DIR+” ; fi”)
```

; same but force the C-shell (csh) to be used

; the single quotes (‘) prevent the Bourne shell from interpreting csh syntax

```
system(“csh -c ‘if (! -d “+DIR+”) then ; mkdir “+DIR+” ; endif’ ”)
```

; execute some local system command

```
system(“msrcp -n ‘mss:/SHEA/REANALYSIS/*’ /ptmp/shea”)
```

UTC = **systemfunc**(“date”) ; use system to get the date/time

fils = **systemfunc** (“cd /some/directory ; ls *nc”)

User-built Functions and Procedures (1 of 4)

- **two ways to load existing files w functions/proc**
 - load "/path/my_script.ncl"
 - use environment variable: NCL_DEFAULT_SCRIPTS_DIR
- **must be loaded prior to use**
 - unlike in compiled language
- **avoid loading functions more than once (undef)**

```
undef ("mult")
function mult(x1,x2,x3,x4)
begin
    return ( x1*x2*x3*x4)
end
```

```
load "/fs/cgd/home0/shea/ncld/mult.ncl"
begin
    x = mult(4.7, 34, 567, 2)
end
```

```
undef ("mult")
function mult(x1,x2,x3,x4)
begin
    return ( x1*x2*x3*x4)
end

begin
    x = mult(4.7, 34, 567, 2)
end
```

User-Built Functions and Procedures (2 of 4)

- Development process similar to Fortran/C
- General Structure:

```
undef ("function_name") ; optional
function function_name (declaration_list)
local local_identifier_list ; optional
begin
    statements
    return (return_value)
end
```

```
undef ("procedure_name") ; optional
procedure procedure_name (declaration_list)
local local_identifier_list ; optional
begin
    statements
end
```

User-Built Functions and Procedures (3 of 4)

- **arguments are passed by reference [fortran]**
- **constrained argument specification:**
 - require specific type, dimensions, and size
 - procedure ex(data[*][*]:float,res:logical,text:string)
- **generic specification:**
 - type only
 - function xy_interp(x1:numeric, x2:numeric)
- **no type, no dimension specification:**
 - procedure whatever (a, b, c)
- **combination**
 - function ex (d[*]:float, x:numeric, wks:graphic, y[2], a)
- **function prototyping**
 - built-in functions are prototyped

User-Built Functions and Procedures (4 of 4)

- additional ('optional') arguments possible
- attributes associated with one or more arguments
 - often implemented as a separate argument (not required)
 - procedure ex(data[*][*]:float, text:string,**optArg:logical**)

```
optArg      = True
optArg@scale = 0.01
optArg@add   = 1000
optArg@wgts  = (/1,2,1/)
optArg@name  = "sample"
optArg@array  = array_3D
ex(x2D, "Example", optArg)
```

```
procedure ex(data, text, opt:logical)
begin
  :
  if (opt .and. isatt(opt,"scale")) then
    d = data*opt@scale
  end if
  if (opt .and. isatt(opt,"wgts")) then
    :
  end if
  if (opt .and. isatt(opt,"array")) then
    xloc3D = opt@array_3D ; nD arrays
  end if           ; must be local before use
end
```

Computations and Meta Data

- **computations can cause loss of meta data**
 - $y = x$; variable to variable transfer; all meta copied
 - $T = T + 273$; T retains all meta data
 - ☞ $T@units = "K"$; user responsibility to update meta
 - $y = 5*x$; y will have no meta data
- **built-in functions cause loss of meta data**
 - $Tavg = \text{dim_avg}(T)$
 - $s = \sqrt{u^2 + v^2}$
- **vinth2p is the exception**
 - retains coordinate variables
 - http://www.cgd.ucar.edu/csm/support/Data_P/vert_interp.shtml
 - hybrid to pressure (sigma to pressure) + other examples

Ways to Retain Meta Data_(1 of 3)

- use copy functions in **contributed.ncl**
 - **copy_VarMeta** (coords + attributes)
 - **copy_VarCoords**
 - **copy_VarAtts**

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
  f = addfile("dummy.nc", "r")
  x = f->X
  ; ----- calculations -----
  xZon = dim_avg (x)
  ; -----copy meta data-----
  copy_VarMeta(x, xZon)
end
```

Ways to Retain Meta Data (2 of 3)

- **use wrapper functions** (eg:)

- dim_avg_Wrap,
- dim_variance_Wrap
- dim_stddev_Wrap
- dim_sum_Wrap
- dim_rmsd_Wrap
- smth9_Wrap
- g2gsh_Wrap
- g2fsh_Wrap
- f2gsh_Wrap
- f2fsh_Wrap
- natgrid_Wrap

- f2fosh_Wrap
- g2gshv_Wrap
- g2fshv_Wrap
- f2gshv_Wrap
- f2fshv_Wrap
- f2foshv_Wrap
- linint1_Wrap
- linint2_Wrap
- linint2_points_Wrap
- eof_cov_Wrap
- eof_cov_ts_Wrap
- zonal_mpsi_Wrap
- etc

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
  f = addfile("dummy.nc", "r")
  x = f->X
  xZon = dim_avg_Wrap(x) ; xZon will have meta data
end
```

Ways to Retain Meta Data_(3 of 3)

- use **variable to variable transfer + dimension reduction** to prefine array before calculation
 - requires that user know **a priori** the array structure

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
    f = addfile("dummy.nc", "r")
    x = f->X                      ; x(time,lev,lat,lon)
; ----- var-to-var transfer + dim reduction-----
    xZon = x(:,:,:,0)              ; xZon(time,lev,lat)
; -----calculations-----
    xZon = dim_avg (x)
    xZon@op = "Zonal Avg: "+x@long_name
end
```

- xZon will have all appropriate meta data of x
- NCL will add an attribute [here: xZon@lon = lon(0)]

Example: read fortran binary, compute psi/chi, output binary

```
begin
    u = fbinrecread ("UV.bin",0, (/120,18,64,128/),"float")
    v = fbinrecread ("UV.bin",1, (/120,18,64,128/),"float")
;-----
; calculate psi and chi as inverse lapacian of divergence
;-----
    psi = ilapsG (uv2vrG(u,v), 0)
    chi = ilapsG (uv2dvG(u,v), 0)
;-----
; output binary data
;-----
    fbinrecwrite ("psichi.bin", -1, psi) ; -1 means append
    fbinrecwrite ("psichi.bin", -1, chi)
end
```

Ex: compute PSI/CHI add meta data

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
    f      = addfile ("UV300.nc", "r")
    u      = f->U
    v      = f->V
; calculate psi and chi
    psi   = ilapsG (uv2vrG(u,v), 0)
    chi   = ilapsG (uv2dvG(u,v), 0)
; copy coordinate variables using function in contributed.ncl
    copy_VarCoords(u, psi)
    copy_VarCoords(u, chi)
; create unique attributes
    psi@long_name = "PSI"
    chi@long_name = "CHI"
    psi@units      = "m2/s"
    chi@units      = "m2/s"
; scale values for plotting
    scale = 1.e06           ; incorporate this into units label
    psi   = psi/scale
    chi   = chi/scale
    .... plot ....
end
```

Convert gaussian=>gaussian, create netCDF, b+w plots

Convert gaussian=>gaussian, create netCDF, b+w plots [cont]

```
dlon = 360./mlon ; longitude spacing for a T42 grid
lon = ispan ( 0,mlon-1,1 )*dlon ; fspan ( 0,(mlon-1)*dlon, mlon) also
lon!0 = "lon"
lon@long_name = "longitude"
lon@units      = "degrees east"

gau_info= doubletofloat(gaus(nlat/2)) ; T42 has 64 lat (64/2=32)
lat = gau_info(:,0) ; gaussian latitudes
lat!0 = "lat"
lat@long_name = "latitude"
lat@units      = "degrees_north"

T42&lat = lat ; add coord var
T42&lon = lon

gwgt= gau_info(:,1) ; gaussian weights
gwgt!0 = "lat"
gwgt@long_name = "gaussian weights"
```

Convert gaussian=>gaussian, create netCDF, b+w plots [cont]

```
system (/usr/bin/rm/ Ex_PR_ex01_T2m_T42.nc") ; remove pre-exist file  
g = addfile ("Ex_PR_ex01_T2m_T42.nc", "c")  
g@title  = "T42 created from T2m.nc"  
g->time  = time  
g->lev   = lev  
g->lat   = lat  
g->lon   = lon  
g->gw    = gwgt  
g->T     = T42  
  
wks  = gsn_open_wks("x11","PR_ex01")          ; open an x11 window  
plot = gsn_csm_contour_map_ce(wks, T63(0,:,:), False) ; default plot  
plot = gsn_csm_contour_map_ce(wks, T42(0,:,:), False) ; default plot  
  
end
```

Regridding via Spherical Harmonics

- some regrid functions use spherical harmonics
- must have global grid to use these functions
- regular functions strip meta data
- wrapper versions preserve attributes **and create coordinate variables**
- no missing data allowed

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
    f      = addfile ("/fs/cgd/data0/shea/CLASS/T2m.nc", "r")
    T63   = f->T
    T42   = g2gsh_Wrap(T63, (/64,128/), 42)
    T25   = g2fgsh_Wrap(T63, (/73,144/) )
end
```

Computations via Spherical Harmonics (1 of 2)

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
  f = addfile ("/cgd/cas/murphys/Data/80.nc", "r")
  u = f->u ; (time,lev,lat,lon)
  v = f->v
  lev = f->lev

  vrt = uv2vrG_Wrap(u,v) ; relative vorticity
  div = uv2dvG_Wrap(u,v) ; divergence

  ud = u ; variable-to-variable copy
  vd = v
  dv2uvg (div, ud, vd) ; divergent wind components
  ud@long_name = "div zonal wind"
  vd@long_name = "div meridionall wind"

  chi = u
  chi = ilapsG (div, 0)
  chi@long_name = "CHI"
  chi@units      = "m2/s"
  psi = u
  psi = ilapsG (vrt, 0)
  psi@long_name = "CHI"
  psi@units      = "m2/s"
```

Computations via Spherical Harmonics (2 of 2)

```
scale = 1.e06 ; scale values for plotting [incorp into label]
chi  = chi/scale ; chi@units = chi@units + "/1e06"
div  = div*scale ; div@units = div@units + "*1e06"
vrt  = vrt*scale ; vrt@units = vrt@units + "*1e06"

wks = gsn_open_wks( wks, "vrdvchi")
gsn_define_colormap( wks, "BIWhRe")

res = True
res@gsnMaximize = True
res@gsnSpreadColors = True
res@cnFillOn = True
res@gsnCenterString = "lev=" + lev({250})

symMinMaxPlt(vrt(0,{250},::), 14, False, res) ; contributed
plt = gsn_csm_contour_map_ce(wks, vrt(0,{250},::),res)
symMinMaxPlt(div(0,{250},::), 14, False, res)
plt = gsn_csm_contour_map_ce(wks, div(0,{250},::),res)

res@cnLinesOn = False
res@gsnLeftString = "Divergent Wind"
res@gsnRightString = chi@long_name
symMinMaxPlt(chi(0,{250},::), 14, False, res)
plt = gsn_csm_vector_scalar_map_ce(wks, ud(0,{250},::),v ud(0,{250},::) \
, chi(0,{250},::), res)
end
```

Calculate Sea Level Pressure from CCM History Tape

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"      ; High Level
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"        ; graphical interfaces
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"    ; user provided util

begin
  MSSPath = "/JHACK/csm/obsst02/ccm3/hist/"
  fili     = "ha221" ; this is a CCM History tape file
  system ("msread "+fili+".ccm "+MSSPath+fili)
  fccm = addfile ("ha221.ccm", "r")

  p0    = 100000.          ; reference pressure (PA)
                          ; p0 = fccm->P0
  hyam = fccm->hyam      ; hybrid coef
  hybm = fccm->hybm
  nhym = dimsizes(hyam)   ; # of hybrid levels [m==>mid]

  ps    = fccm->PS         ; sfc pressure [time,lat,lon]
  phis = fccm->PHIS(0,:,:); sfc geopotential[time,lat,lon]
  T    = fccm->T(:,:,nhym-1,:); read T at lowest sig lvl [time,lat,lon]
  pres = p0*hyam(nhym-1) + hybm(nhym-1)*ps ; pres at lowest sigma level
  psl  = pslec(T, phis, ps, pres) ; sea-level pressure (Pa) [time,lat,lon]
  copy_VarCoords(ps,psl)         ; shea_misc
  psl@long_name = "sea level pressure"
  psl@units       = "Pa"
```

Calculate Sea Level Pressure from CCM History Tape [cont]

```
pslAve = dim_avg (psl(lat|:,lon|:,time|:))      ; time average all the psl [lat,lon]
pslStd  = dim_stddev (psl(lat|:,lon|:,time|:))   ;std dev of all psl          [lat,lon]
                                         ; not shown (coord var and attribute 'stuff')

wks = gsn_open_wks ("x11", "PR_ex03")

psl  = psl*0.01           ; convert to hPa [mb]
psl@units = "hPa"        ; change units attribute

res  = True                ; user wishes to change from default
res@cnLevelSpacing = 5.0 ; force 5 hPA contour interval

do nt=0,dimsizes(fccm->time)-1 ; plot each time step
    plot = gsn_csm_contour_map_ce (wks, psl(nt,:,:),res)
end do

end
```

Example: Arbitrary Transect

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

begin
    ; open file and read in data
    diri = "/fs/scd/home1/shea/ncldata_input/"
    fili = "01-50.nc"
    f = addfile(diri+fili , "r")
    T = f->T                      ; T(time,lev,lat,lon)
    ; create arrays of lat and lon points
    lonx = (/ 295, 291.05, 287.4 , 284, 281, 274.5, 268, 265 /)
    laty = (/ -30, -25.2, -20.3, -15, -10.3, 0.0, 9.9, 15 /)

    ; interpolate data to given lat/lon points
    transect = linint2_points_Wrap (T&lon, T&lat, T, True,lonx,laty, 0)
    ; transect(time,lev, 8)
end

ncl < PR_ex04.ncl
```

Ex: Climatology, Seasonal Average, EOFs

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
begin
    f      = addfile ("data_in.nc", "r")
    P      = f->prc
    ntime = dimsizes(P&time)
; calculate monthly climatology and monthly standard deviation
    Pclm  = climMonTLL(P)
    Pstd  = stdMonTLL (P)
; reorder array to calculate running average
    Psea  = runave (P(lat|:,lon|:,time|:), 3, 1 )
    Psea @time_op = "3-month running mean: seasonal ave"
; area weight
    wlat  = sqrt(cos(0.01745329*Psea&lat) )
    wPsea= Psea                                     ; transfer meta data
    wPsea= wPsea*conform(wPsea, wlat, 0)          ; wgt at all grid pts
; calculate EOFs
    eof    = eofcov_Wrap (wPsea(:,:,0:ntime-1:12), 3)
    eofTs = eofcov_ts_Wrap (Psea(:,:,0:ntime-1:12) , eof)
end
```

Empirical Orthogonal Functions (EOFs)

- **principal components, eigenvector analysis**
- **provide efficient representation of variance**
 - May/may not have dynamical information
- **successive eigenvalues should be distinct**
 - if not, the eigenvalues and associated patterns are noise
 - 1 from 2, 2 from 1 and 3, 3 from 2 and 4, etc
 - North et. al (*MWR*, July 1982: eq 24-26) provide formula
- **geophysical variables: spatial/temporal correlated**
 - no need sample every grid point
 - ☞ no extra information gained
 - ☞ oversampling increases size of covar matrix + compute time
- **patterns are domain dependent**

EOFs via NCL

- **eofcov** (**eofcov_pcmsg**) when **variances** are important
- **eofstd** (**eofstd_pcmsg**) when **patterns** are important
 - data are standardized **prior** to calculations
- **eofcov_ts** and **eofstd_ts**
 - create time series of pattern amplitudes
- **eof_varimax**
 - rotate EOFs using Kaiser varimax criterion
 - currently no % variance calculated
- Use **_Wrap** versions in **contributed.ncl**

```
let x(time,lat,lon) and X = x(lat |: , lon |:, time |:)  
  - wgt grid points => X = X*conform(X, wgt_lat, 0)  
eof      = eofcov_Wrap (X, neof)                                ; eofcov(...)  
eofJan  = eofcor_Wrap (X(:, :, 0:ntim-1:12), neof)      ; every 12th  
eofJul  = eofcov_pcmsg_Wrap(X(:, :, 6:ntim-1:12), neof, 80)  
e_ts    = eofcov_ts_Wrap(x(lat |: , lon |:, time |:) , eof)
```

SVDs via NCL

- **svd_lapack** – general rectangular matrix
- **svdcov (svdstd)** – Bretherton-Smith-Wallace
- **svdcov_sv (svdstd_sv)** – return left/right sing vectors

Toy Processor (1 of 3)

- Specify variables, Derived quantities, Options [sketch]

```
load "..."  
begin  
    fileCreate      = True           ; create output file  
    plotColor       = True           ; color plots  
    plotBw          = True           ; black/white plots  
    plotType        = "ncgm"         ; ps or x11  
    fields          = (/ "T", "SPEED" /) ; specify output  
    inFils          = systemfunc ("ls *.nc") ; files to be used  
; ----- end user input  
  
    nFlds = dimsizes( fields )      ; # desired fields  
    nFils = dimsizes( inFils )       ; # desired files  
  
    if (fileCreate) then            ; create output file  
        fo  = addfile ("toyOutput.nc", "c")  
    end if  
    if (plotBw) then                ; create B+W  
        wksBw = gsn_open_wks (plotType, "toyOutputBw")  
    end if  
    if (plotColor) then             ; create Color output  
        wksCol = gsn_open_wks (plotType, "toyOutputCol")  
    end if
```

Toy Processor (2 of 3)

```
do nFld=0,nFlds-1                                ; loop over desired fields
  do nFil=0,nFils-1                               ; loop over files
    f      = addfile (inFils(nFil), "r")          ; current input file
    Var   = getfilevarnames (f)                     ; variables on the file
    nVar= dimsizes (Var)                           ; number of variables

    if (any(fields(nFld).eq.Var)) then            ; is it a model field??
      ;OrigVarProcedure (Var(nFld),fileCreate,plotBw,plotColor) ; some procedure
      continue                                     ; go to next iteration
    end if

    if (field(nFld).eq."SPEED") then              ; is SPEED desired
      if (isvar("U") .and. isvar("V")) then        ; are U and V in memory
        SPEED = sqrt( U^2 + V^2 )
      else
        if (isfilevar(f,"U") .and. isfilevar(f,"V")) then ; are U and V on the file
          U = f->U
          V = f->V
          SPEED = sqrt( U^2 + V^2 )
        else
          print ("SPEED can not be calculated")      ; U/N not available
          continue                                     ; go to next iteration
        end if
      end if
    end if
```

Toy Processor (3 of 3)

```
copy_VarAttrs ( U , SPEED) ; copy attributes
copy_CoordVars ( U , SPEED) ; copy coordinate variables
SPEED@long_name = "speed"

if (fileCreate) then
    fo->SPEED = SPEED           ; file output
end if
if (plotBw) then
    ;plot function (wksBw, ...)   ; B&W plot
end if
if (plotColor) then
    ;plotfunction (wksCol, ...)   ; Color plot
end if
continue                      ; go to next iteration
end if                          ; end if SPEED
end do                          ; end nFil
end do                          ; end nFld

end
```

Better to use "**addfiles**" and process all files at once? depends on user needs

External Fortran-C Codes

external codes, Fortran, C, or local commerical libraries may be executed from within NCL

- **generic process**

- develop a **wrapper** (interface) to transmit arguments
- compile the external code (eg, f77, f90, cc)
- Link the external code to create a **shared object**

- **specifying where shared objects are located**

- external statement
 - ☞ **external "/dir/code.so"**
- system environment variable:
 - ☞ **LD_LIBRARY_PATH**
- NCL environment variable:
 - ☞ **NCL_DEF_LIB_DIR**
 - ☞ external functions need not have **::** before use

Create/Use a Fortran Shared Object (1 of 2)

Step 1

- bracket subroutine + argument declarations with **interface delimiters**
- only **codes actually called** from NCL need special interface delimiters
- no Fortran argument can be named **data** (bug)

C NCLFORTSTART

```
subroutine foo ( xin,xout, mlon, nlat, text)
integer mlon, nlat
real xin(mlon,nlat), xout(mlon,nlat)
character(*) text
```

C NCLEND

rest of fortran code; may include many subroutines or other declarations

Create/Use a Fortran Shared Object (2 of 2)

Step 2: create shared object using WRAPIT utility

- WRAPIT foo.f
- WRAPIT foo.stub foo.f90

Step 3: add external statement to NCL script

- external SO_NAME "path_name"
 - ☞ SO_NAME is arbitrary (capital by convention)
 - ☞ external DEMO "./foo.so" (".".so" by convention)

Step 4: invoking the shared object in the script

- SO_NAME::fortran_name(arguments)
- DEMO::foo(x,y,m lon,n lat,label)

what WRAPIT does

- automatically creates NCL-fortran interface(s)
- uses wrapit77 to create C interface [f77 syntax]
 - wrapit77 < foo.f >! foo_W.c
- only uses code enclosed between delimiters
 - input can be code fragment(s) or full subroutine(s)
- compiles and creates object modules
 - nhlcc -c foo_W.c ↗ foo_W.o
 - nhlf90 -c foo.f ↗ foo.o
- creates dynamic shared object [.so] using ld
 - SGI: ld -64 -o foo.so -shared foo_W.o foo.o -fortran
 - SUN: /usr/ccs/bin/ld -o foo.so foo_W.o foo.o -G -lf90 -L /opt/SUNWspro/lib -l sunperf
- removes extraneous intermediate files

WRAPIT –h <return> will show options and examples

NCL/Fortran Argument Passing

- **arrays: NO reordering required**
 - $x(\text{time}, \text{lev}, \text{lat}, \text{lon}) \ Leftrightarrow x(\text{lon}, \text{lat}, \text{lev}, \text{time})$

- **ncl: $x(N,M) \ Leftrightarrow x(M,N)$:fortran [M=3, N=2]**
 - $x(0,0) \ Leftrightarrow x(1,1)$
 - $x(0,1) \ Leftrightarrow x(2,1)$
 - $x(0,2) \ Leftrightarrow x(3,1)$
 - $x(1,0) \ Leftrightarrow x(1,2)$
 - $x(1,1) \ Leftrightarrow x(2,2)$
 - $x(1,2) \ Leftrightarrow x(3,2)$

- **numeric types must match**
 - integer \Leftrightarrow integer
 - double \Leftrightarrow double
 - float \Leftrightarrow real

- **Character-Strings: a nuisance [C,Fortran]**

Example: Linking in Fortran

STEP 1: quad.f

C NCLFORTSTART

```
subroutine cquad(a,b,c,nq,x,quad)
dimension x(nq), quad(nq)
```

C NCLEND

```
do i=1,nq
    quad(i) = a*x(i)**2 + b*x(i) + c
end do
return
end
```

C NCLFORTSTART

```
subroutine prntq (x, q, nq)
integer nq
real x(nq), q(nq)
```

C NCLEND

```
do i=1,nq
    write (*,"(i5, 2f10.3)") i, x(i), q(i)
end do
return
end
```

STEP 2: quad.so

WRAPIT quad.f

STEPS 3-4

```
external QUPR  "./quad.so"
begin
    a = 2.5
    b = -.5
    c = 100.
    nx = 10
    x = fspan(1., 10., 10)
    q = new (nx, float)
    QUPR::cquad(a,b,c, nx, x,q)
    QUPR::prntq (x, q, nx)
end
```

Example: Linking F90 routines

```
STEP 1: quad90.stub
C NCLFORTSTART
subroutine cquad (a,b,c,nq,x,quad)
dimension x(nq), quad(nq) ! ftn default
C NCLEND
C NCLFORTSTART
subroutine prntq (x, q, nq)
integer nq
real x(nq), q(nq)
C NCLEND
```

```
prntq_I.f90
module prntq_I
interface
subroutine prntq (x,q,nq)
real, dimension(nq) :: x, q
integer, intent(in) :: nq
end subroutine end interface
end module

cquad_I.f90
module cquad_I
interface
subroutine cquad (a,b,c,nq,x,quad)
real, intent(in) :: a,b,c
integer, intent(in) :: nq
real,dimension(nq), intent(in) :: x
real,dimension(nq),intent(out) :: quad
end subroutine end interface
end module
```

```
quad.f90
subroutine cquad(a, b, c, nq, x, quad)
implicit none
integer , intent(in) :: nq
real , intent(in) :: a, b, c, x(nq)
real , intent(out) :: quad(nq)
integer :: i ! local
quad = a*x**2 + b*x + c ! array
return
end subroutine cquad
```

```
subroutine prntq(x, q, nq)
implicit none
integer , intent(in) :: nq
real , intent(in) :: x(nq), q(nq)
integer :: i ! local
do i = 1, nq
    write (*, '(I5, 2F10.3)') i, x(i), q(i)
end do
return
end
```

```
STEP 2: quad90.so
WRAPIT quad90.stub printq_I.f90 \
cquad_I.f90 quad.f90
```

```
STEP 3-4: same as previous
ncl < PR_quad90.ncl
```

Fortran vs. NCL: string - character

NCL (C) ↗↗ Fortran: string/character interchange problematical

C NCLFORTSTART

```
subroutine csdemo (string_in, string_out)
    character*(*) string_in ! passed FROM ncl
    character*8   string_out ! passed TO ncl
```

C NCLEND

```
print *, str_in
str_out = "F-to-NCL"
return
end
```

external DEMO "./csdemo.so"

begin

```
cstring = new (8, character)
```

```
DEMO::csdemo("NCL-F" , cstring)
```

```
stringc = chartostring( cstring )
```

```
print ( stringc )
```

end

Can **NOT** pass **arrays** of strings or characters

Example: Linking IMSL (NAG,...) routines

```
STEP 1: rcurvWrap.f
C NCLFORTSTART
subroutine rcurvwrap (n, x, y, nd, b, s, st, n1)
integer n, nd, n1
real x(n), y(n), st(10), b(n1), s(n1)
C NCLEND
call rcurv(n,x,y,nd,b,s,st)      ! IMSL
return
end
```

STEP 2: rcurvWrap.so

WRAPIT -I mp -L /usr/local/lib64/r4i4 -l imsl_mp rcurvWrap.f

```
external IMSL "./rcurvWrap.so"
begin
  x = (/ 0,0,1,1,2,2,4,4,5,5,6,6,7,7 /)
  y = (/508.1, 498.4, 568.2, 577.3, 651.7, 657.0, 755.3 \
        758.9, 787.6, 792.1, 841.4, 831.8, 854.7, 871.4 /)

  nobs = dimsizes(y)
  nd   = 2
  n1   = nd+1
  b    = new ( n1, typeof(y))
  s    = new ( n1, typeof(y))
  st   = new (10, typeof(y))

  IMSL::rcurvwrap(nobs, x, y, nd, b, s, st, n1)
end
```

Combining NCL and Fortran in C-shell

```
#!/usr/bin/csh
# ===== NCL =====
cat >! main.ncl << "END_NCL"
load "/fs/cgd/data0/shea/nclGSUN/gsn_code.ncl"
load "/fs/cgd/data0/shea/nclGSUN/gsn_csm.ncl"
load "/fs/cgd/data0/shea/nclGSUN/contributed.ncl"
external SUB "./sub.so"
begin
...
end
"END_NCL"
# =====FORTRAN =====
cat >! sub.f << "END_SUBF"
C NCLFORTSTART
...
C NCLEND
"END_SUBF"
# ===== WRAPIT=====
WRAPIT sub.f
# ===== EXECUTE =====
ncl main.ncl >! main.out
```

using NCL like a scripting language

```
begin
  mssi = getenv ("MSSOCNHIST") ; get environment variable
  diri = "/ptmp/user/"           ; dir containing input files
  fili = "b20.007.pop.h.0"       ; prefix of input files
  diro = "/ptmp/user/out/"      ; dir containing output files
  filo = "b20.TEMP."            ; prefix of output files

  nyRStrt = 300                  ; 1st year
  nyRLast= 999                   ; last year
  do nyear=nyRStrt,nyRLast
    print ("---- "+nyear+" ----")
                                ; read 12 months for nyyear
    msscCmd = "msrcp -n 'mss:' +mssi+ fili+nyear+ "-[0-1][0-9].nc' "+diri+"."
    print ("msscCmd="+msscCmd)
    system (msscCmd)
                                ; strip off the TEMP variable
    ncocCmd = "ncrcat -v TEMP " +diri+filii+".nc "+ diro+filo+nyear+".nc"
    print ("ncocCmd="+ncocCmd)
    system (ncocCmd)
                                ; remove the 12 monthly files
    rmcCmd = "rm' "+diri+filii+nyear+.nc"
    print ("rmcCmd="+rmcCmd)
    system (rmcCmd)
  end do
end
```